12

ESD-TR-82-403, Vol. I

MTR-8278, Vol. I

# TRANSFORM DECODING OF REED-SOLOMON CODES VOLUME I: ALGORITHM AND SIGNAL PROCESSING STRUCTURE

By
D. O. CARHOUN
B. L. JOHNSON
S. J. MEEHAN

NOVEMBER 1982

Prepared for

SOLID STATE SCIENCES DIVISION
ROME AIR DEVELOPMENT CENTER
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts

Project No. 7010/7170
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-81-C-0001

## REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

Jerry Silverman
JERRY SILVERMAN
Project Engineer

Harold Roth
HAROLD ROTH, Director
Solid State Sciences Division

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>ESD-TR-82-403, Vol. I | 2. GOVT ACCESSION NO.<br>AD-A123953 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>TRANSFORM DECODING OF REED-SOLOMON CODES<br>VOLUME I: ALGORITHM AND SIGNAL PROCESSING STRUCTURE | | 5. TYPE OF REPORT & PERIOD COVERED<br><br>6. PERFORMING ORG. REPORT NUMBER<br>MTR-8278, Vol. I |
| 7. AUTHOR(s)<br>D. O. CARHOUN, B. L. JOHNSON, S. J. MEEHAN | | 8. CONTRACT OR GRANT NUMBER(s)<br>F19628-81-C-0001 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>The MITRE Corporation<br>Burlington Road<br>Bedford, MA 01730 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>Project No. 7010/7170 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Solid State Sciences Division<br>Rome Air Development Center<br>Hanscom AFB, MA 01731 | | 12. REPORT DATE<br>November 1982<br>13. NUMBER OF PAGES<br>66 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED<br>15a. DECLASSIFICATION DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
ERROR LOCATION
REED-SOLOMON ENCODING AND DECODING
SIGNAL PROCESSING
TRANSFORM DECODING

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses in detail a transform decoding algorithm and its hardware implications, for the Reed-Solomon codes, that offer major simplifications relative to the conventional BCH decoding algorithm. A fast algorithm for encoding and syndrome computation is described. Modification of the error location process to accommodate erasures is also described. Also discussed are hardware implementation issues with a summary of design features and parameters to be incorporated in a future set of programmable integrated circuits for decoding a large number of Reed-Solomon codes.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(*When Data Entered*)

UNCLASSIFIED

SECURITY CLASSIFICATION OF TH[illegible] PAGE(*When Data Entered*)

## PREFACE

The second printing of Volume I of this report coincides with the publication of ESD-TR-82-403, Vol. II, Transform Decoding of Reed-Solomon Codes Volume II: Logical Design and Implementation. The revised printing of the first volume contains no new material other than corrections of typographical errors.

## ACKNOWLEDGMENTS

This document has been prepared under Project 7010/7170, Contract F19628-81-C-0001. The contract is sponsored by the Solid State Sciences Division, Rome Air Development Center, Hanscom Air Force Base, Massachusetts.

## TABLE OF CONTENTS

| Section | | Page |
|---|---|---|
| | LIST OF ILLUSTRATIONS | 5 |
| | LIST OF TABLES | 6 |
| I | INTRODUCTION | 7 |
| | 1.1 Purpose | 7 |
| | 1.2 Background | 8 |
| | 1.3 Scope | 10 |
| II | TRANSFORM APPROACH | 12 |
| | 2.1 Linear Block Codes | 12 |
| | 2.2 Maximum Distance Codes | 12 |
| | 2.3 Reed-Solomon Codes | 13 |
| III | REED-SOLOMON TRANSFORM ENCODING | 20 |
| | 3.1 Reed-Solomon Encoding | 20 |
| | 3.2 Codeword Generation by Discrete Transformation | 20 |
| | 3.3 A Fast Transform Algorithm | 24 |
| | 3.4 Field-Programmability of the Fast Transform Algorithm | 28 |
| | 3.4.1 Multi-Dimensional Transform Partitioning | 29 |
| IV | REED-SOLOMON TRANSFORM DECODING ALGORITHM | 37 |
| | 4.1 Syndrome Computation | 37 |
| | 4.2 Error Locator | 39 |
| | 4.2.1 The Berlekamp-Massey Algorithm | 40 |
| | 4.2.2 Modification for Correction of Errors and Erasures | 44 |
| | 4.2.3 Continued Fraction Algorithm | 47 |
| | 4.3 Message Extraction | 51 |

TABLE OF CONTENTS (CONCLUDED)

| Section | | Page |
|---|---|---|
| V | HARDWARE DESIGN (SUMMARY) | 52 |
| | 5.1 Functional Partitioning and Programmability | 52 |
| | 5.2 Throughput | 58 |
| | REFERENCES | 61 |

## LIST OF ILLUSTRATIONS

| Figure | | Page |
|---|---|---|
| 1 | Conventional BCH Decoding Algorithm | 16 |
| 2 | Transform Decoding Algorithm | 18 |
| 3 | Structure for Computing the Transform by Polynomial Evaluation | 23 |
| 4 | Structure for Computing a 31-Point Transform over $GF(2^5)$ | 26 |
| 5 | Structure for Computing a 255-Point Transform over $GF(2^8)$ using Two-Dimensional Partitioning | 35 |
| 6 | Canonic Ladder For Stieltjes Continued Fraction Realization ($\nu=3$) | 49 |
| 7 | Transform Section | 53 |
| 8 | Error Locator | 54 |

## LIST OF TABLES

| Table | | Page |
|---|---|---|
| I | Comparison of Computational Complexity | 27 |
| II | Transforms over GF($2^m$) | 29 |
| III-1 | Minimal Irreducible Polynomials over GF($2^8$) | 31 |
| III-2 | Minimal Irreducible Polynomials over GF($2^7$) and GF($2^6$) | 32 |
| III-3 | Minimal Irreducible Polynomials over GF($2^5$) and GF($2^4$) | 33 |
| IV | Reed-Solomon Codes Accommodated by the Hardware Design | 55 |
| V | Half Rate Reed-Solomon Codes Accommodated by the Hardware Design | 56 |
| VI | Decoding Times for Half-Rate Reed-Solomon Codes | 59 |

## SECTION I

## INTRODUCTION

### 1.1 Purpose

One of the continuing concerns of the Low Cost Electronics project is the application of new technology to the implementation of error-correcting codes for reliable data communication. Our interest stems from the need for low-cost hardware to implement error-correction codes that exhibit significant coding gain on interference-resistant communication channels. Previous studies of coding gain led us to concentrate work on the implementation of the Reed-Solomon class of generalized BCH codes. This class of codes, although well-suited to the correction of both isolated random errors and random error bursts because of its optimum distance properties, continues to be genuinely in need of efficient decoding algorithms implemented by low-cost hardware.

This report examines a transform decoding algorithm and its hardware implications for the Reed-Solomon codes. This algorithm offers major simplifications relative to the more conventional BCH decoding algorithm, resulting from reduced algorithm complexity and from the opportunity to apply fast computational techniques for implementing its major portions.

While processing functions required for algebraic coding have been associated conventionally with digital _data_ processing, the basic functions of the transform decoding method are convolution and discrete transformation, typical _signal_ processing functions. An essential difference between the error coding requirements and the conventional digital signal processing practice is that the former require the operations to be carried out in a finite algebraic field or ring. Although this territory may seem unfamiliar to the digital signal processing community, in actuality many familiar concepts apply directly, and in some instances implementation may be simpler (e.g., elimination of carry bits, multiplication by adding "logarithms") if the unique properties of the finite structure are exploited. Although it will become evident that the error coding functions impose a relatively high degree of complexity on the processing circuitry, it is hoped that the analogies with more conventional linear digital signal processing functions will aid in the development of functional LSI hardware - and perhaps also lead to the use of finite-field methods where useful and appropriate for other linear signal processing purposes.

## 1.2 Background

Previous work on the low-cost electronics project demonstrated that error correction coding, when used in conjunction with spread-spectrum modulation, provides an added dimension to the design of jam-resistant communications systems. Error correction codes are useful to correct message errors caused by interference, additive random noise, and other disturbance present on the channel. They do so by adding redundant symbols according to a predetermined strategy (coding) and then extracting the correct message from the noisy received signal (decoding), aided by prior knowledge of the code. Given sufficent channel signal-to-noise ratio (measured at the receiver) it can be shown that this represents an effective means of obtaining low error probability in the decoded message, even though the energy per transmitted symbol is reduced because of the added redundancy (we assume a fixed energy available per source message symbol). In other words, the coding strategy produces an effective gain (at a modest price in bandwidth) that more than compensates for the necessity to spread the available energy over the combined message and redundancy symbols. The net result is a coding gain obtained by computational processing that may range from a few dB for channels corrupted by additive gaussian noise to some tens of dB for randomly fading noisy channels. As the tolerance to errors in the output data is lowered, the arguments for using error coding are strengthened.

Under this project we previously examined the application of the BCH decoding algorithm for decoding Reed-Solomon codes with particular attention to its structural implementation. We also experimented with the direct decoding of Reed-Solomon codes having short block lengths by implementing a code-table search algorithm under microprocessor control. The work reported here is concerned with the algorithm and structures for decoding Reed-Solomon codes by application of a transform method.

If the message symbols to be encoded by a Reed-Solomon code lie in the domain of a linear Fourier-like transformation from the message to the sequence that is transmitted over the communication channel, then it is possible to eliminate a number of computational steps in the decoding algorithm. The finite-field linear transforms involved are directly analogous to the discrete Fourier transform pair defined over the complex number field, except that in this case they are defined in the finite algebraic field that contains the code symbols. The transform algorithm can be summarized by the following steps:*

Encoding

1. Define a sequence in the algebraic domain of the channel code transform as the message symbols accompanied by a sequence of zeros.

*The transform method of decoding Reed-Solomon codes has a historical precedent in the definition of the codes, but only recently has there been renewed interest in its application [1], [2], [3].

2. Generate the channel codeword by taking the forward transform of the defined sequence. (This produces a non-systematic channel code.)

Decoding

1. Take the inverse transform of the received sequence.
2. Isolate the error syndrome from the inverse transform and use it to determine the error locator polynomial or equivalently the connection polynomial of a corresponding linear feedback shift register (LFSR).
3. Extrapolate the inverse error-transform sequence using the unforced response of the synthesized LFSR, and subtract it from the inverse transform of the received sequence. The difference is the corrected message sequence.

In comparison with the conventional BCH decoding algorithm as applied to Reed-Solomon codes, the transform decoding algorithm eliminates the need to search for roots of the error locator polynomial. It is also unnecessary to calculate the error values in the co-domain of the channel code, a step that in effect solves a set of simultaneous linear equations. For a non-systematic channel code, the BCH decoding algorithm requires polynomial division by the code generator polynomial of the corrected received sequence in order to retrieve the information symbols. This step is eliminated in the transform algorithm because the message symbols are defined in the domain that contains the inverse error transform values.

## 1.3 Scope

The next section of this report describes in overview the Reed-Solomon codes and the transform decoding algorithm in a convenient algebraic setting in which the discrete transform and its inverse are regarded as equivalent to polynomial evaluation and interpolation. In Section III a fast algorithm for implementing the transform and its application for the encoder are discussed. Section IV presents the major steps of the decoding algorithm, including modification of the basic algorithm to accommodate the correction of both errors and erasures up to the minimum distance limit of the code.

In Section V we discuss some hardware implementation issues and present a summary of recommended design features and parameters to be incorporated in a set of integrated circuits to enable decoding of a large number of Reed-Solomon codes. The second volume of this report [4] documents the detailed logical design of circuitry and the construction of a discrete logic breadboard.

## SECTION II

## THE TRANSFORM APPROACH

### 2.1 Linear Block Codes

A linear block code is a linear mapping (homomorphism) from a k-dimensional vector space to a larger n-dimensional vector space; the redundancy is n-k. The message space may consist, for example, of $(2^m)^k$ sequences -- or k-tuples -- of m-bit message symbols while the code space consists of $(2^m)^n$ code sequences - or n-tuples - of m-bit symbols containing $(2^m)^k$ code sequences that combine both message and redundancy. The mapping increases the distance (or spacing) between code words corresponding to message sequences, distance being measured as the number of different symbols between pairs of such code words. In fact this distance increases exponentially as the redundancy increases linearly. When random errors occur in the transmitted code words, the increased spacing permits decoding by a maximum distance (equivalent to maximum-likelihood) decoding rule. When the block length of the code becomes sufficiently long, such a decoding procedure becomes impractical. One may then resort to the algebraic properties of the code construction to facilitate practical decoding.

### 2.2 Maximum Distance Codes

The Reed-Solomon codes in which we are interested are members of a class of linear block codes called maximum distance separable codes. These are codes for which the minimum distance between pairs of code words equals the maximum value

$$d = n - k + 1 \qquad (1)$$

admitted by the Hamming volume (or sphere-packing) bound. For such codes it is possible to correct a number of symbol errors not exceeding one-half the redundancy of the code. Arguments based on the triangle inequality can be used to show that twice as many erasures are correctable, an erasure (rather than an error) being defined as a codeword symbol of unknown (rather than incorrect) value occurring at a known (rather than unknown) symbol location. In fact, it can be shown that it is possible to correct any combination of t errors and s erasures provided that the inequality

$$2t + s \leq n - k \tag{2}$$

is satisfied for the maximum distance codes.

In order to discuss the structural properties of such codes and the apparatus for coding and decoding, it is convenient to describe an n-tuple of m-bit symbols by a polynomial of degree n-1 having coefficients that are members of the finite algebraic field of $2^m$ elements. Such a polynomial is determined uniquely by its n coefficients, or equivalently by its values at any n distinct points of the field. A code word of block length n, for example, may be specified either by a set of n values or by the polynomial coefficients interpolated from those values.

2.3 Reed-Solomon Codes

A maximal distance code of block length n and dimension k can be generated by a set of k functions $\{g_i(z)\}$ defined over an algebraic field F containing at least n elements provided that the functions in the set are independent and that no linear combination of them has more than k-1 roots in the field. The Reed-Solomon codes are a special case for which the function set is $\{1, z, z^2, z^3, \ldots, z^{k-1}\}$[5]. We are concerned here with codes (and messages) for which the symbols are binary m-tuples. These are elements of the Galois field of $2^m$ elements. The elements of this finite field can be written as $0, b, b^2, \ldots, b^{2^m-1}$, where b is a primitive element of the field. For the Reed-Solomon codes, let the message be represented by the polynomial $f(x) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{k-1} z^{k-1}$ where the coefficients $a_i$ are elements of the field $F = GF(2^m)$ that represent the set of k message symbols. The mapping from the message space to the code space is accomplished by evaluating the message polynomial f(x) at the $2^m-1$ non-zero points of the field. The transmitted code word consists of the sequence of $n = 2^m-1$ values

$$f(b),\ f(b^2), \ldots,\ f(b^{2^m-1})$$

resulting from the polynomial evaluation. To decode the message after receiving the code sequence, one may form the n equations

$$\begin{aligned} f(b) &= a_0 + a_1 b + a_2 b^2 + \ldots + a_{k-1} b^{k-1} \\ f(b^2) &= a_0 + a_1 b^2 + a_2 b^4 + \ldots + a_{k-1} b^{2k-2} \\ &\cdot \\ &\cdot \\ &\cdot \\ f(b^{2^m-1}) &= a_0 + a_1 + a_2 + \ldots + a_{k-1} \end{aligned} \tag{3}$$

where the coefficients $a_i$ are the unknown values to be found by solution of the equation set. Any subset of k of these equations is linearly independent and may therefore be used. If one or more of the code symbols are received in error, then different solutions may result from the various k-subsets. The errors may be corrected by solving all of the distinct k-subsets and taking a majority vote. Such an approach, although valid in principle, is generally not practical to implement directly because of the large number of equation sets to be solved, each implying the inversion of a square matrix of dimension k.

Combinatorial methods based on (n,k,t)-covering systems have been devised as a way of trapping the correctable error patterns in order to reduce systematically the number of equation sets to be solved. When combined with a minimum distance decoding rule, the covering method can yield practical results for codes of modest block length. It can also accommodate shortened cyclic codes.

For codes of moderately long block length, one can successfully apply a concise decoding algorithm based on the algebraic properties and structure of the code. The well-known BCH decoding algorithm depends on calculating an error syndrome as a linear transformation of the received channel symbols and then using the syndrome to determine an error locator polynomial as the solution of a linear recursion (the so-called key equation) [6]. The roots of the error-locator, which may be found by a systematic root-search, designate the locations of the errors (or the multiplicative inverses of the locations). These values are then used to solve a set of linear equations for the channel errors, enabling correction of the channel code and subsequent extraction of the message [7].

The error locator is the key step of the algorithm. It may be regarded as a means of determining which set of k equations drawn from the full set of equation (3) is sufficient to solve for the message symbols, since having determined which values $f(b^i)$ are in error, the remaining ones must be hypothesized to be correct. The conventional BCH decoding algorithm is described schematically in Figure 1.

In our work we resort to a transform decoding algorithm for the Reed-Solomon codes that is a variation of the BCH decoding algorithm which allows a major reduction in the computational complexity of the algorithm. The principal features of our implementation include:

1. Interpolation of the code word sequence to produce a coefficient syndrome.
2. Solution of a linear recursion based on the interpolated coefficients.
3. Extrapolation of a sequence of error coefficients to extract the message coefficients.

It will be shown that the first step is equivalent to taking an n-point transform (defined over the finite field $GF(2^m)$) that is

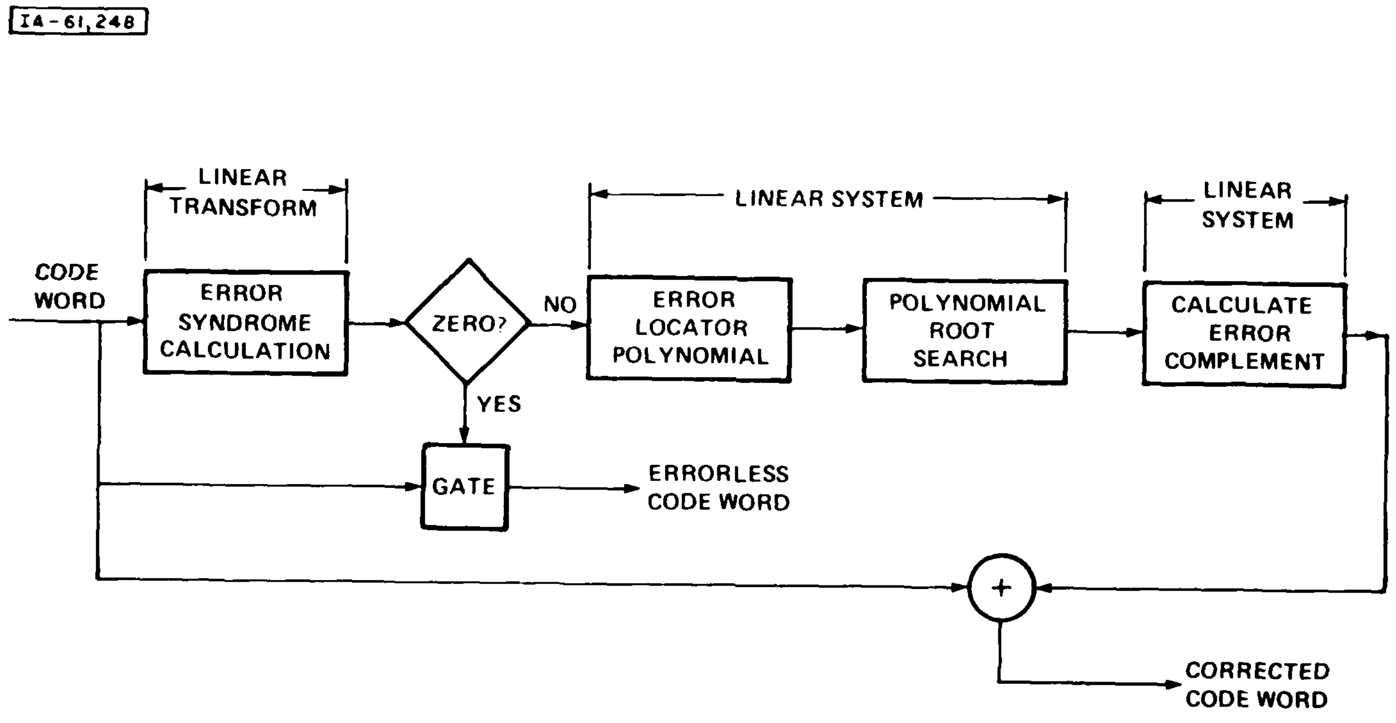


Figure 1. Conventional BCH Decoding Algorithm

completely analogous to the discrete Fourier transform defined over the field of complex numbers. The second step, which determines the symbol error locations, uses an algorithm (Berlekamp-Massey) that synthesizes the shortest linear feedback shift register that generates a prescribed sequence [8]. Each iteration of this algorithm involves a convolution of the syndrome coefficients with the shift register tap weights. The third step utilizes the unforced response of the synthesized LFSR, again involving convolution. A simple block diagram of the decoding apparatus is shown in Figure 2.

The transform approach to Reed-Solomon codes is evident in their original exposition [1], the connection between the message symbols and the channel code being variously regarded as based on the Mattson-Solomon polynomial [9], the Chinese remainder theorem [10], LaGrange's interpolation formula [11], and more recently number-theoretic Fourier-like transforms [2,4,12,13]. The computational advantages of a transform approach to encoding and decoding relative to the more or less standard BCH decoding algorithm have been previously discussed [2,4,12,13] but efficient hardware implementing these codes is still not commonly available. The principal computational advantages are (1), the ability to both encode and compute the error syndrome with fast algorithms that mimic FFT algorithms, and (2), the ability to predict the transform of the channel error pattern from the error-locator polynomial. The second advantage avoids a root-search for the error-locations, followed by generation of the error evaluation polynomial and explicit computation of the channel error values. Instead, to decode one needs only to: a) compute the transform of the received channel sequence, the error syndrome constituting a subset of the transform, b) compute

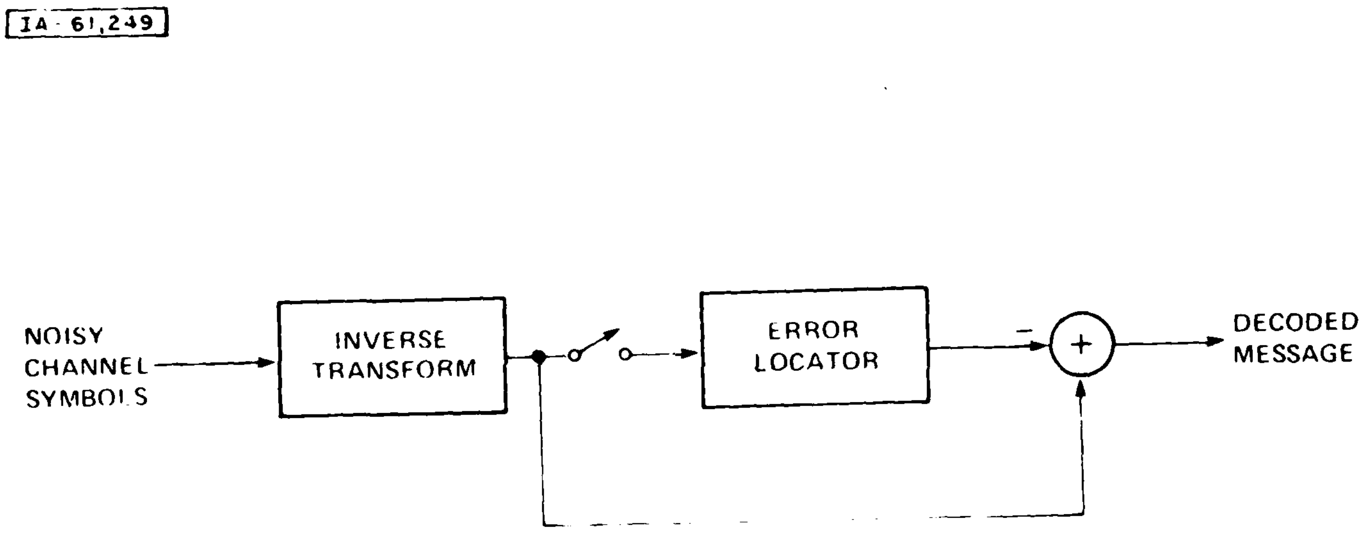


Figure 2. Transform Decoding Algorithm

the error locator polynomial by an iterative algorithm (Berlekamp-Massey algorithm or continued-fractions) which operates on the syndrome, and c) generate the transform of the error sequence as a linear recursion with the error locator polynomial. The validity of the third step was proved by Reed, et al. [2].

## SECTION III

## REED-SOLOMON TRANSFORM ENCODING

### 3.1 Reed-Solomon Encoding

Reed-Solomon codes were first described in 1960 [2]. Shortly afterwards they were generalized and a decoding algorithm based on their algebraic properties was suggested [6]. Today these codes are usually described as the polynomial product of a generator function g(z) defined over a finite field GF(q) with an information sequence polynomial defined over the same field [7]. In algebraic terminology, the code forms an ideal in the ring of polynomials modulo $z^n - 1$ over GF(q) that is generated by g(z). The roots of g(z), which are contained in GF(q), consist of a consecutive set of elements of the cyclic multiplicative group of GF(q). Since the code ideal consists of all products of g(z), each member when evaluated at any of the roots of g(z) becomes congruent to zero. This is the basis for determining the error syndrome of the channel sequence which is required for further decoding. Below we discuss the generation of codewords from the transform viewpoint.

### 3.2 Codeword Generation by Discrete Transformation

We have described the function of code generation in Section II as n-point evaluation of a message polynomial defined over a finite algebraic field. The inverse function is interpolation of the polynomial from its n values. These functions (evaluation and interpolation) are inverse in the sense that their (commutative) product is the identity function. It has also been shown that these functions form a "transform" pair analogous to the discrete Fourier transform pair defined over the complex number field [14]. In our case the transforms are defined over the finite algebraic field of the code.

Let $a_0$, $a_1$, . . ., $a_{n-1}$ be distinct elements of a finite algebraic field GF($p^m$) of order $p^m - 1$, having an element b of order n. The linear transformation

$$A_j = \sum_{i=0}^{n-1} a_i b^{ij} \tag{4}$$

is an endomorphic mapping of GF($p^m$). It is assumed that n divides $p^m - 1$, the order of the field, and for our purposes $n = p^m - 1$. In that case the field element b is a primitive nth root of unity. It can be shown that for any integer r,

$$\sum_{i=0}^{n-1} b^{ir} = \begin{cases} n, & r \equiv 0 \bmod n \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

and the property can be used to verify by direct calculation that the mapping that is inverse to that of equation (4) is the linear transformation

$$a_i = n^{-1} \sum_{j=0}^{n-1} A_j b^{-ji} \tag{6}$$

where $-n^{-1}n = p^m - 1$. Equations (4) and (6) define a discrete transform pair over GF($p^m$) and the operations of addition and multiplication are defined in the same field. Addition may be performed as modulo-p addition of the m-tuples that are the field elements composing the sum. Multiplication may be defined by addition of indices of the field elements

$$b^r \, b^s = b^{(r + s) \bmod p^m - 1} \tag{7}$$

The transform pair of equation (4) and equation (6) is analogous to the discrete Fourier transform pair for which b would be a complex nth root of unity and the arithmetic would be defined in the complex number field. The conceptual value of the Fourier transform pair is preserved in the finite field. In particular, the cyclic convolution property holds. Fast computational algorithms, analogous to the FFT algorithms, can also be applied.

If the sequence to be transformed is expressed as a polynomial over $GF(p^m)$

$$a(z) = a_o + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1} \qquad (8)$$

then the transform of the sequence $a_o, a_1, a_2, \ldots, a_{n-1}$ is identical with polynomial evaluation of $a(z)$ at the $n$ distinct points $b^o, b^1, b^2, \ldots, b^{n-1}$ and the inverse transform is identical with interpolation of the polynomial $a(z)$ from its $n$ values.

In order to generate a Reed-Solomon $(n,k)$ code, we let the set $a_o, a_1, \ldots, a_{k-1}$ represent the message symbols, setting $a_i = 0$ for $i = k, k+1, \ldots, n-1$, and evaluate the resulting polynomial (by calculating the transform of the message sequence) at the $n = p^m - 1$ non-zero units of the field. The transform, or polynomial evaluation, can be expressed as a continued product

$$a(b^j) = a_o + b^j (a_1 + \ldots + b^j (a_{n-2} + b^j a_{n-1}) \ldots) \qquad (9)$$

or equivalently it can be interpreted as the remainder of the polynomial division $a(z) / (z-b^j)$ evaluated at $b^j$. The second interpretation may be represented as the set of polynomial congruences

$$a(b^j) \equiv a(z) \bmod (z-b^j), \; j = 0, 1, \ldots, n-1. \qquad (10)$$

A structure for computing the transform is shown in Figure 3. Notice that the same structure can be used for calculating both the transform and its inverse. The method shown requires $n^2$ separate products in $GF(p^m)$ to be formed by sequencing $n$ symbols through a set of registers containing $n$ separate multipliers. Since the hardware

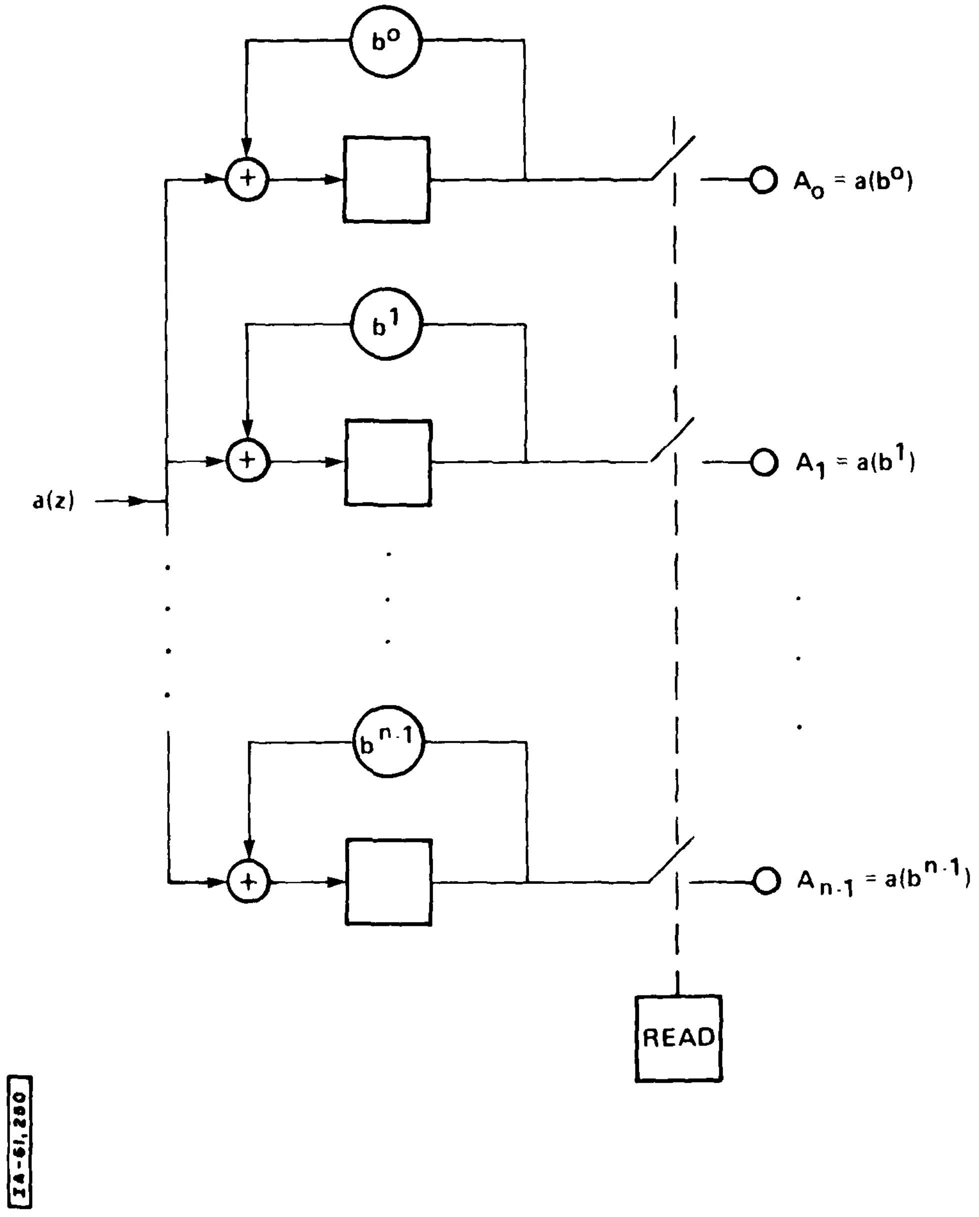


Figure 3. Structure for Computing the Transform By Polynomial Evaluation

complexity of finite field multiplication is formidable, we should look for "fast" computational algorithms that reduce the number of multiplications in $GF(p^m)$. Alternatively, one could use table-look-up multiplication at the sacrifice of speed.

## 3.3 A Fast Transform Algorithm

A fast transform algorithm -- one that tends to minimize the number of multiplications in $GF(p^m)$ -- can be devised to compute the transform pair of equation (4) and equation (6). The set of congruences of equation (10) can be calculated in principle by dividing the polynomial $a(z)$ separately by the first degree polynomials $(z-b^j)$, keeping only the remainders. That is operationally equivalent to evaluating $a(z)$ at the n non-zero field points $b^j$. In either case $n^2$ multiplications in $GF(p^m)$ are implied.

An equivalent method is to first divide $a(z)$ by a smaller set of polynomials of higher degree containing distinct factors of the form $(z-b^j)$, and then to evaluate the remainder polynomials at the appropriate values $b^j$. If this set of divider polynomials is the set of minimal polynomials of the non-zero field elements, then their coefficients are elements of the prime field GF(p) so that only scalar multiplication by the elements of the prime field is required in the first step. This is particularly significant when p = 2 and the corresponding scale factors are either zero or one. The equivalence of the two methods is easily seen by examination of the factorization over $GF(p^m)$ of the polynomial $z^n - 1$:

$$z^n - 1 = \prod_{j=0}^{p^m-1} (z - b^j); \qquad b \in GF(p^m) \qquad (11)$$

which can also be expressed as the product of the minimal polynomials $m_i(z)$ having as roots the value $b^i$ and its conjugates $(b^i)^{p^\ell}$. This factorization is given explicitly by

$$z^n - 1 = \prod_{i=1}^{M} m_i(z) \qquad (12a)$$

where M is the number of irreducible factors of $z^n - 1$ and

$$m_i(z) = \prod (z-b^i)\,(z-b^{ip})\;.\;.\;.\;(z-b^{ip^q}). \qquad (12b)$$

The minimal polynomials $m_i(z)$ have their coefficients restricted to the prime field GF(p).

To complete the transform computation, we must evaluate each of the remainders at the conjugate roots of the associated minimal polynomial divisor. This second step requires multiplications in GF($p^m$), but the number is substantially reduced because there is a relatively small number of remainder polynomials, each of smaller degree than the degree of field extension.

As an illustration of the fast algorithm, a structure for computation of a 31-point transform over GF($2^5$) is shown in Figure 4. Notice that division by the 5th degree minimal polynomial factors of $z^{31}$-1 is accomplished in a set of six binary feedback shift registers of length m = 5. The additional transform point corresponding to division by the first degree polynomial factor z-1 is calculated simply by summing the coefficients of a(z). In the structure shown we use a single multiplier in GF($2^5$) sequentially to evaluate the remainder polynomials. This choice of implementation is used in the hardware design of our encoder and decoder because of the relative complexity of such a multiplier and also because it supplies data at a sufficient rate for the remaining operations. For the example given, the number of multiplications in GF($2^5$) resulting from the remainder evaluation has been reduced from 930 to 120 by the use of this algorithm. The corresponding reductions for some other situations of interest are listed in Table I.

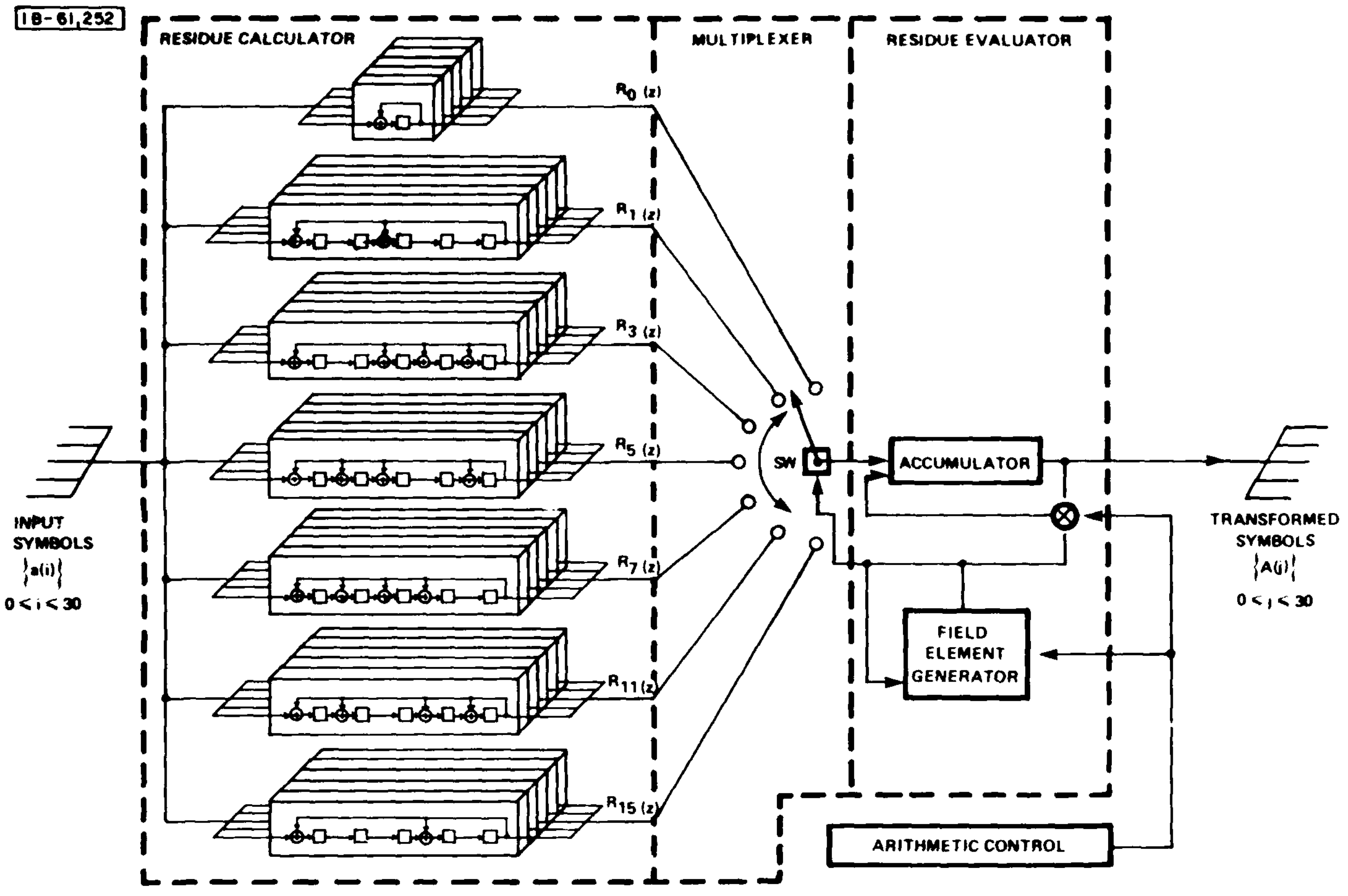


Figure 4. Structure for Computing a 31-Point Transform over $GF(2^5)$

Table I: Comparison of Computational Complexity

| Transform Size N | *Field of Operation* | Number of Extension Field Multiplications | |
|---|---|---|---|
| | | Conventional Transform | Fast Transform |
| 255 | GF($2^8$) | 64,770 | 1,720 |
| 127 | GF($2^7$) | 16,002 | 756 |
| 85 | GF($2^8$) | 7,140 | 572 |
| 63 | GF($2^6$) | 3,906 | 284 |
| 51 | GF($2^8$) | 2,550 | 338 |
| 31 | GF($2^5$) | 930 | 120 |
| 21 | GF($2^6$) | 420 | 74 |
| 17 | GF($2^8$) | 272 | 112 |
| 15 | GF($2^4$) | 210 | 38 |
| 9 | GF($2^6$) | 72 | 32 |
| 7 | GF($2^6$) | 42 | 12 |

## 3.4 Field Programmability of the Fast Transform Algorithm

The hardware for the fast transform algorithm described in section 3.3 can be reconfigured easily to compute transforms of appropriate lengths in the different Galois fields $GF(2^m)$. In each field of order $2^m$ a transform of n points can be defined for every integer n that divides $2^m -1$. Thus in $GF(2^8)$, transforms of 255, 85, 51, 17, 5 and 3 points and the trivial 1-point (identity) transform can be defined. Division by the minimal polynomial factors that split $z^n -1$ is performed in a set of binary feedback shift registers of length m. There are m of these registers identically configured, operating on the m-bit symbols of the input sequence, and used in association with each of the minimal polynomial factors. As an example, for a 31-point transform over $GF(2^5)$, there are six 5th-degree factors and one first-degree factor that split $z^{31}-1$. Division by each polynomial factor requires 5 binary feedback shift registers wired with the same connection polynomial, as shown in Figure 4.

We have designed our Reed-Solomon coder and decoder to accommodate codes, of both maximum and sub-maximum lengths, having symbol fields ranging from 4 to 8 bits per symbol. The lengths of the non-trivial transforms defined in the various fields are tabulated in Table II together with the minimal polynomial factors of $z^n -1$ for the different transform lengths n in each field. The minimal polynomials are given in Table III-1 through Table III-3. If we provide for a 255-point transform over $GF(2^8)$; by reconfiguring the lengths and connection polynomials of the feedback shift registers, all of the other cases can be accommodated. Of course, the multiplier in $GF(2^m)$ that is used in evaluating the remainders needs also to be reconfigured. The design details of the hardware that can be electronically reconfigured to compute the full set of transforms enumerated in Table II are described in Volume II of this report. We have constructed a breadboard, with medium scale logic, of a transformer used in a Reed-Solomon

TABLE II

Transforms over GF($2^m$)

| Field of Calculation | Transform Length N | Required Minimal Polynomial Divisors |
|---|---|---|
| GF($2^8$) | 255 | $m_0(z)$, $m_1(z)$, $m_3(z)$, $m_5(z)$, $m_7(z)$, $m_9(z)$, $m_{11}(z)$, $m_{13}(z)$, $m_{15}(z)$, $m_{17}(z)$, $m_{19}(z)$, $m_{21}(z)$, $m_{23}(z)$, $m_{25}(z)$, $m_{27}(z)$, $m_{29}(z)$, $m_{31}(z)$, $m_{37}(z)$, $m_{39}(z)$, $m_{43}(z)$, $m_{45}(z)$, $m_{47}(z)$, $m_{51}(z)$, $m_{53}(z)$, $m_{55}(z)$, $m_{59}(z)$, $m_{61}(z)$, $m_{63}(z)$, $m_{85}(z)$, $m_{87}(z)$, $m_{91}(z)$, $m_{95}(z)$, $m_{111}(z)$, $m_{119}(z)$, $m_{127}(z)$ |
|  | 85 | $m_0(z)$, $m_3(z)$, $m_9(z)$, $m_{15}(z)$, $m_{21}(z)$, $m_{27}(z)$, $m_{39}(z)$, $m_{45}(z)$, $m_{51}(z)$, $m_{63}(z)$, $m_{87}(z)$, $m_{111}(z)$ |
|  | 51* | $m_0(z)$, $m_5(z)$, $m_{15}(z)$, $m_{25}(z)$, $m_{45}(z)$, $m_{55}(z)$, $m_{85}(z)$, $m_{95}(z)$ |
|  | 17 | $m_0(z)$, $m_{15}(z)$, $m_{45}(z)$ |
|  | 15* | $m_0(z)$, $m_{17}(z)$, $m_{51}(z)$, $m_{85}(z)$, $m_{119}(z)$ |
|  | 5 | $m_0(z)$, $m_{51}(z)$ |
|  | 3* | $m_0(z)$, $m_{85}(z)$ |
| GF($2^7$) | 127 | $m_0(z)$, $m_1(z)$, $m_3(z)$, $m_5(z)$, $m_7(z)$, $m_9(z)$, $m_{11}(z)$, $m_{13}(z)$, $m_{15}(z)$, $m_{19}(z)$, $m_{21}(z)$, $m_{23}(z)$, $m_{27}(z)$, $m_{29}(z)$, $m_{31}(z)$, $m_{43}(z)$, $m_{47}(z)$, $m_{55}(z)$, $m_{63}(z)$ |

* Cases included in breadboard

TABLE II (Concluded)

Transforms over GF($2^m$)

| Field of Calculation | Transform Length N | Required Minimal Polynomial Divisors |
|---|---|---|
| GF($2^6$) | 63 | $m_0$ (z), $m_1$ (z), $m_3$ (z), $m_5$ (z), $m_7$ (z), $m_9$ (z), $m_{11}$ (z), $m_{13}$ (z), $m_{15}$ (z), $m_{21}$ (z), $m_{23}$ (z), $m_{27}$ (z), $m_{31}$ (z) |
| | 21* | $m_0$ (z), $m_3$ (z), $m_9$ (z), $m_{15}$ (z), $m_{21}$ (z), $m_{27}$ (z) |
| | 9 | $m_0$ (z), $m_7$ (z), $m_{21}$ (z) |
| | 7* | $m_0$ (z), $m_9$ (z), $m_{27}$ (z) |
| | 3* | $m_0$ (z), $m_{21}$ (z) |
| GF($2^5$) | 31* | $m_0$ (z), $m_1$ (z), $m_3$ (z), $m_5$ (z), $m_7$ (z), $m_{11}$ (z), $m_{15}$ (z) |
| GF($2^4$) | 15* | $m_0$ (z), $m_1$ (z), $m_3$ (z), $m_5$ (z), $m_7$ (z) |
| | 5* | $m_0$ (z), $m_3$ (z) |
| | 3* | $m_0$ (z), $m_5$ (z) |

* Cases included in breadboard

Table III - 1

Minimal Irreducible Polynomials over $GF(2^8)$

$$m_i(z) = m_0 + m_1 z^1 + m_2 z^2 + m_3 z^3 + m_4 z^4 + m_5 z^5 + m_6 z^6 + m_7 z^7 + m_8 z^8$$

| Polynomial | $m_o$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ |
|---|---|---|---|---|---|---|---|---|---|
| $m_0$ (z) | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $m_1$ (z) | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| $m_3$ (z) | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| $m_5$ (z) | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| $m_7$ (z) | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| $m_9$ (z) | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| $m_{11}$ (z) | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| $m_{13}$ (z) | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $m_{15}$ (z) | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| $m_{17}$ (z) | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $m_{19}$ (z) | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| $m_{21}$ (z) | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| $m_{23}$ (z) | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $m_{25}$ (z) | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| $m_{27}$ (z) | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| $m_{29}$ (z) | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| $m_{31}$ (z) | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| $m_{37}$ (z) | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| $m_{39}$ (z) | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| $m_{43}$ (z) | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| $m_{45}$ (z) | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| $m_{47}$ (z) | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $m_{51}$ (z) | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| $m_{53}$ (z) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| $m_{55}$ (z) | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| $m_{59}$ (z) | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| $m_{61}$ (z) | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| $m_{63}$ (z) | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| $m_{85}$ (z) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $m_{87}$ (z) | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| $m_{91}$ (z) | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| $m_{95}$ (z) | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| $m_{111}$(z) | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| $m_{119}$(z) | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $m_{127}$(z) | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

Table III - 2

Minimal Irreducible Polynomials over GF($2^7$)

$m_i(z) = m_0 + m_1 z^1 + m_2 z^2 + m_3 z^3 + m_4 z^4 + m_5 z^5 + m_6 z^6 + m_7 z^7$

| Polynomial | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ |
|---|---|---|---|---|---|---|---|---|
| $m_0$ (z) | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $m_1$ (z) | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| $m_3$ (z) | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| $m_5$ (z) | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| $m_7$ (z) | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $m_9$ (z) | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| $m_{11}$ (z) | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $m_{13}$ (z) | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| $m_{15}$ (z) | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| $m_{19}$ (z) | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| $m_{21}$ (z) | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| $m_{23}$ (z) | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| $m_{27}$ (z) | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| $m_{29}$ (z) | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $m_{31}$ (z) | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $m_{43}$ (z | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| $m_{47}$ (z) | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| $m_{55}$ (z) | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| $m_{63}$ (z) | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

Minimal Irreducible Polynomials over GF($2^6$)

$m_i(z) = m_0 + m_1 z^1 + m_2 z^2 + m_3 z^3 + m_4 z^4 + m_5 z^5 + m_6 z^6$

| Polynomial | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ |
|---|---|---|---|---|---|---|---|
| $m_0$ (z) | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| $m_1$ (z) | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| $m_3$ (z) | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| $m_5$ (z) | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| $m_7$ (z) | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $m_9$ (z) | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $m_{11}$ (z) | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| $m_{13}$ (z) | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| $m_{15}$ (z) | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| $m_{21}$ (z) | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| $m_{23}$ (z) | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| $m_{27}$ (z) | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| $m_{31}$ (z) | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

Table III - 3

Minimal Irreducible Polynomials over GF($2^5$)

$m_i(z) = m_0 + m_1 z^1 + m_2 z^2 + m_3 z^3 + m_4 z^4 + m_5 z^5$

| Polynomial | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ |
|---|---|---|---|---|---|---|
| $m_0$ (z) | 1 | 1 | 0 | 0 | 0 | 0 |
| $m_1$ (z) | 1 | 0 | 1 | 0 | 0 | 1 |
| $m_3$ (z) | 1 | 0 | 1 | 1 | 1 | 1 |
| $m_5$ (z) | 1 | 1 | 1 | 0 | 1 | 1 |
| $m_7$ (z) | 1 | 1 | 1 | 1 | 0 | 1 |
| $m_{11}$ (z) | 1 | 1 | 0 | 1 | 1 | 1 |
| $m_{15}$ (z) | 1 | 0 | 0 | 1 | 0 | 1 |

Minimal Irreducible Polynomials over GF($2^4$)

$m_i(z) = m_0 + m_1 z^1 + m_2 z^2 + m_3 z^3 + m_4 z^4$

| Polynomial | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ |
|---|---|---|---|---|---|
| $m_0$(z) | 1 | 1 | 0 | 0 | 0 |
| $m_1$(z) | 1 | 1 | 0 | 0 | 1 |
| $m_3$(z) | 1 | 1 | 1 | 1 | 1 |
| $m_5$(z) | 1 | 1 | 1 | 0 | 0 |
| $m_7$(z) | 1 | 0 | 0 | 1 | 1 |

encoder and decoder. It will compute transforms of up to 51 points in $GF(2^8)$ and the transforms in the smaller fields, as indicated by asterisks in Table II. The hardware implementation of the breadboard is also described in Volume II.

### 3.4.1 Multi-dimensional Transform Partitioning

It has been shown elsewhere that a transform over $GF(2^m)$ of n points can be decomposed into a multi-dimensional transform having one-dimensional components of transform lengths $n_1$, $n_2$, . . $n_k$, where the individual lengths are a complete set of relatively prime factors of n [15]. In that work, the individual one-dimensional transforms were computed by means of the Winograd algorithm for fast cyclic convolution. The multi-dimensional transform decomposition can also be combined with our algorithm for fast polynomial evaluation to produce a fast transform algorithm with attractive hardware implications. For example, a 255-point transform over $GF(2^8)$ could be configured as a 3-dimensional transform of component lengths of 17, 5 and 3 points respectively. Equivalently it could be configured as a 2-dimensional transform of 17 and 15 points, or of 85 and 3 points, or of 51 and 5 points. Consider the last case: the full transform can be configured as 5 transforms of 51 points each followed by 51 transforms of 5 points each. The 51-point transforms would use identical (or time-shared) hardware but operate on different input data, selected as every 5th term of the input sequence appropriately offset. The 5-point transforms are calculated in succession from the 5 outputs of the 51-point transforms. This example is of direct interest to us in the use of our 51-point transform breadboard. Expansion to 255 points using this 2-dimensional decomposition merely requires replication of the 51-point transform hardware, followed by a single 5-point transformer used sequentially. A block diagram of such a configuration is shown in Figure 5. In practice, a larger number of 5-point transformers would be used (perhaps 5) in order to maintain throughput.

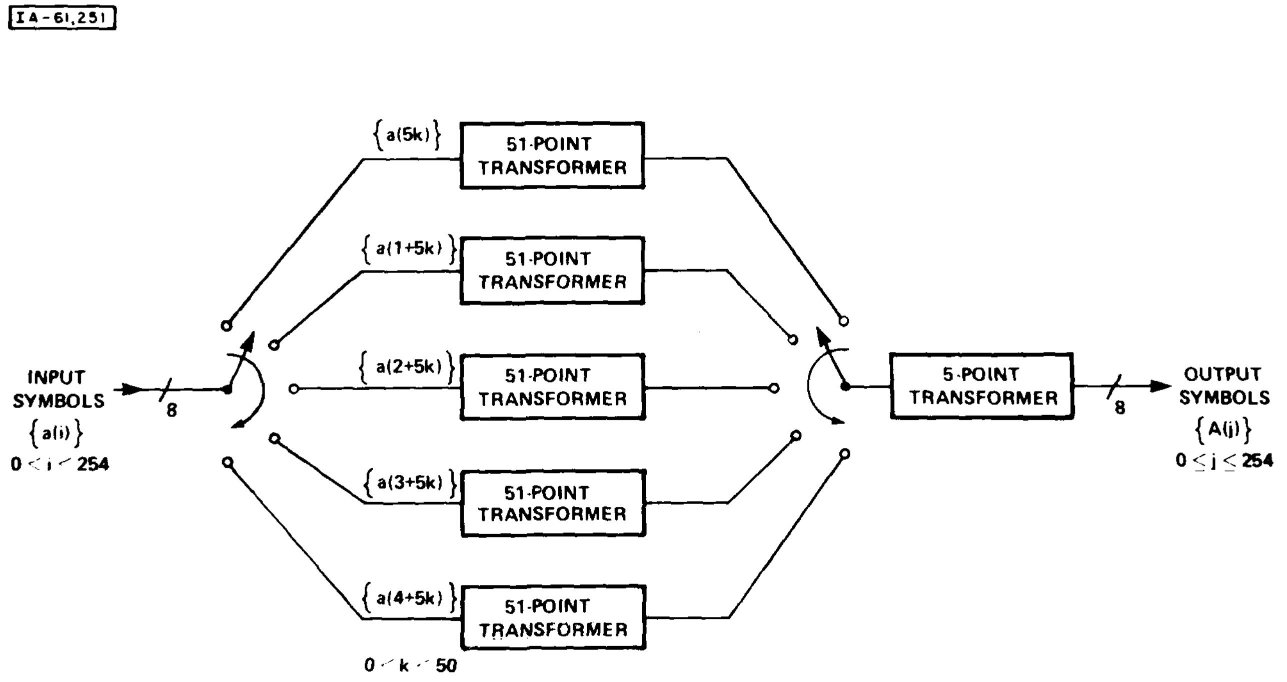


Figure 5. Structure for Computing a 255-Point Transform Over $GF(2^8)$ Using Two-Dimensional Partitioning

Comparison of the multi-dimensional approach with a direct method of transformation, both based on our algorithm for fast polynomial evaluation, will show that approximately the same amount of hardware (polynomial dividers, field multipliers, logic gates) is needed in either case. But in the multi-dimensional case there is greater opportunity for the use of replicated hardware. In the example shown in Figure 5 there are 35 polynomial division registers, but only 8 different types rather than 33 different types required for the direct implementation. There are 255 multipliers in GF($2^m$) required in the example, but only 56 different types. In general a multi-dimensional transform of length $n = \prod_{i=1}^{k} n_i$ points requires polynomial evaluation to be performed at only $\ell = \sum_{i=1}^{k} n_i$ different roots.

## SECTION IV

## REED-SOLOMON TRANSFORM DECODING ALGORITHM

Assume that a message represented by the polynomial a(z) is encoded as a Reed-Solomon codeword by the transform, or polynomial evaluation, method described in Section III and that a number of errors, within the constraint of equation (2), occurs in transmission. The received code sequence contains these errors. If we know which symbols are correctly received, (which is the same as knowing which symbols are in error) we can choose a correct subset of k linearly independent equations from the full set of equation (3) and solve them for the message symbols. An equivalent procedure, and the one that is most often used, is to find the error locations in the received sequence and then to use them directly to solve for the error values, correcting the received code sequence from which the message can be decoded. Since there are usually fewer errors than message symbols, the second method should require less computational effort. As discussed in Section II, we will use a transform version of the algebraic decoding algorithm to streamline the required operations. But we must first determine the error syndrome from which the error locator polynomial may be determined.

### 4.1 Error Syndrome Computation

Let the source message be represented by the polynomial $a(z) = a_0 + a_1z + a_2z^2 + . . . + a_{k-1}z^{k-1}$, which we will regard as having degree n-1 but with the n-k highest-degree coefficients equal to zero. The transmitted codeword is represented by the polynomial $A(z) = A_0 + A_1z + A_2z^2 + . . . + A_{n-1}z^{n-1}$ in which the coefficient $A_j$ is determined as $a(b^j)$ in accordance with the transform of equation (4). If we were to apply the inverse transform of equation (6) to the coefficient sequence of A(z), we would obtain the message a(z).

Notice in particular that for any valid $A(z)$ the interpolated message coefficients $a_i$ must equal zero for $i \geq k$. This construction is equivalent to choosing the set $\{b^{-k}, b^{-(k+1)}, b^{-(k+2)}, \ldots, b^{-n+1}\}$ as the roots of the polynomial $g(z)$ that generates the code ideal in the ring of polynomials modulo $z^n - 1$.

Assume that an error sequence represented by the polynomial $E(z) = E_0 + E_1 z + E_2 z^2 \ldots + E_{n-1} z^{n-1}$ has been added to the encoded message upon transmission. If the received word is correctable, then $E(z)$ will have no more than $\frac{n-k}{2}$ non-zero coefficients; both their values and locations will be unknown. The received sequence is represented by the polynomial sum $R(z) = E(z) + A(z)$ so its inverse transform is $r(z) = e(z) + a(z)$, where $e(z)$ is the inverse transform of $E(z)$ and $a(z)$ is the original message. The decoding problem is to determine $e(z)$ in order to extract the message $a(z)$ from the inverse transform $r(z)$ of the observed sequence $R(z)$.

To compute the error syndrome, we first interpolate the polynomial $r(z)$ from the values of the received sequence $R(z)$ by taking its inverse transform,

$$r_i = n^{-1} \sum_{j=0}^{n-1} R_j b^{-ji} \tag{14}$$

which is equivalent to multiplying by the constant factor $n^{-1}$ the values $R(b^{-i})$ that result from evaluation of the received sequence polynomial $R(z)$. Since $b^{-i} \equiv b^{n-i}$ it follows that the transform structures described in Section III can also be used to calculate the inverse transform, provided that we index the output values in reverse order and multiply them by the scale factor $n^{-1}$. The codes that concern us here have symbols that are binary m-tuples, and consequently $n = 2^m - 1 \equiv 1 \bmod 2^m$ so that the scale factor is unity, all of which will be assumed below.

Since $A(b^{-i}) \equiv 0$ for $i \geq k$ because of the method of code construction, we can separate from equation (14) a term valid for i=k, k + 1, . . ., n-1,

$$s_i = R(b^{-i}) = \sum_{j=0}^{n-1} E_j b^{-ji}; \; i=k,\ k+1,\ \ldots,\ n-1. \qquad (15)$$

The sequence $\{s_i\}$ of the n-k values calculated from equation (15) constitutes the error syndrome associated with the channel error pattern E(z). This error syndrome is, by definition, equal exactly to the last n-k values of the inverse transform of the received channel sequence, given by equation (14). The values of equation (14) for i < k are, in general, different from the values that we would obtain by extending the definition of equation (15).

## 4.2 Error Location

After the error syndrome has been calculated, it can be used to determine the locations of the errors in the channel error pattern E(z). Several procedures are available for determining the error locations, including a method of continued fractions [16], another method that applies Euclid's algorithm [5], and an algorithm formulated by Berlekamp and Massey [8]. Common to all of these methods is the determination of a polynomial whose distinct roots designate the error locations (or their multiplicative inverses).

In our work, we employ the Berlekamp-Massey algorithm, judging it to be an available method that is computationally efficient and conceptually satisfactory from a signal processing viewpoint. It may be regarded as an algorithm for synthesizing the shortest linear feedback shift register that generates a prescribed sequence obtained from the inverse transform (interpolation) of the received code sequence. The error locator polynomial is the characteristic polynomial of the LFSR; its coefficients uniquely satisfy a linear recursion with the first n-k coefficients of the interpolated sequence. In our use of this algorithm, we include correction for erasures by initializing the algorithm in accordance with the known erasure locations. The linear recursion satisfied by the LFSR is simply a convolution between the input sequence and the feedback coefficients.

A competing method, one that may be simpler conceptually although not as convenient for hardware implementation with linear sequential circuits, is based on a continued-fraction development of a power series expansion of the error syndrome. The continued fraction can be used to synthesize directly a canonic ladder realization of an equivalent rational polynomial transfer function, or its values can be used to compute by iteration the characteristic polynomial of the LFSR. Both methods will be described.

### 4.2.1 The Berlekamp-Massey Algorithm

The Berlekamp-Massey algorithm has been discussed thoroughly by its authors [6, 8]. The discussion presented here, which is taken in part from a previous project document, is included merely for the sake of completeness and for continuity with the transform decoding method being described.

The channel error sequence $E(z)$ is described by a list of pairs of field elements, $Y_i$ (the value of the error) and $X_i$ (an error location determined by the index of a field element) for the $i^{th}$ symbol error [7]. The syndrome values may be expressed in terms of these elements as

$$R(b^{-j}) = s_j = \sum_{i=1}^{\nu} Y_i X_i^{\,j}; \; j=k,\ k+1,\ .\ .\ .,\ n-1 \qquad (16)$$

(in accordance with the code construction of Section III) where $\nu$ is the Hamming weight of the error pattern or equivalently the number of non-zero coefficients of $E(z)$. We assume $\nu \leq (n-k)/2$ so that the error bound of the code is not exceeded.

The algorithm is concerned with determining the coefficients of an error-locator polynomial

$$\sigma(z) = \prod_{i=1}^{\nu} (z-X_i) = \sigma_\nu + \sigma_{\nu-1}z + \sigma_{\nu-2}z^2 + \ldots + z^\nu \qquad (17)$$

whose distinct roots are the error locations. The effect of introducing the error-locator polynomial is to reduce the system of non-linear equations relating the error values, locations, and syndrome components to a system of separate sets of linear equations.
In the BCH decoding algorithm, a system of equations is first solved for the error locations $X_i$, which reduces equation (16) to a linear system relating the syndrome values $s_j$ and error values $Y_i$. The error locator polynomial provides an intermediate step in the process that is useful for determining the error locations $X_i$.

For the transform decoding algorithm, the error locator polynomial has even greater significance. First, a unique linear relationship can be established between the syndrome values and the coefficients of the error locator polynomial, namely

$$s_j \sigma_\nu + s_{j+1} \sigma_{\nu-1} + \ldots + s_{j+\nu-1}\sigma_1 + s_{j+\nu} = 0 \qquad (18)$$

which is valid for all $k \leq j \leq n-1-\nu$. This relationship is established by multiplying (both sides) of equation (17) by $Y_iX_i^j$, then substituting $X_i$ for z, summing the result over the index $1 \leq i \leq \nu$, and substituting from equation (16). Next, it can be proven that there exists a polynomial e(z) of degree less than n that satisifies linear recursion (18) with $\sigma(z)$ for all $1 \leq j \leq n$, and that e(z) is uniquely specified by $\nu$ consecutive values of its transform $E(b^{-j})$, (where for example $j=k, k+1, \ldots, k+\nu-1$) and E(z) has no more than $\nu$ non-zero coefficients.* In that case E(z) is the channel error pattern, and the calculated values $E(b^{-j})$ form its inverse transform.

*The proof is given in Appendix A of reference [2].

The error locator polynomial can be determined from the error syndrome and then be used to extrapolate the inverse transform e(z), which must be subtracted from the inverse transform r(z) of the received channel sequence to extract the message. The values $E(b^{-j})$ for $j \geq k$ exactly balance the syndrome coefficients to produce zeros in those message positions. There is no need to explicitly find the error locations in order to solve a set of linear equations for the channel error values, as is usually done in the BCH decoding algorithm.

The Berlekamp-Massey algorithm provides an iterative method of synthesizing the canonic linear feedback shift register that has the characteristic polynomial $\sigma(z)$ which is used to extrapolate the inverse error transform e(z). The algorithm is in fact a constructive proof that if the length $L_r$ and connection polynomial $\sigma^{(r)}(z)$ are known for the minimal length LFSR that generates the sequence $(s_k, s_{k+1}, \ldots, s_r)$ but <u>not</u> the sequence $(s_k, s_{k+1}, \ldots, s_r, s_{r+1})$, then a valid choice for the connection polynomial to generate the latter sequence is

$$\sigma^{(r+1)}(z) = z^{r-m}\sigma^{(r)}(z) - d_r d_m^{-1}\sigma^{(m)}(z) \qquad (19)$$

where the next discrepancy $d_r$ is defined as

$$d_r = s_{r+k} + \sum_{i=1}^{L_r} \sigma_i^{(r)} s_{r+k-i} \qquad (20)$$

and the maximum degree of $\sigma^{(r+1)}(z)$, which is also the minimum length of the shift register is

$$L_{r+1} = \begin{cases} \max(L_r, r+1-L_r) & ; d_r \neq 0 \\ L_r, & ; d_r = 0. \end{cases} \qquad (21)$$

The recursion begins with the initial conditions established for $r=0, -1$,

$$\begin{aligned} L_{-1} &= 0 & L_0 &= 0 \\ \sigma^{(-1)} &= 1 & \sigma^{(0)} &= 1 \qquad (22) \\ d_{-1} &= 1 & d_0 &= s_k. \end{aligned}$$

The index r corresponds to the rth step in the recursion of equation (19); it is also the length of the correct sequence generated by the minimal shift register of length $L_r$ when $d_r=0$. The index m is the index of $\sigma^{(m)}(z)$, the last connection polynomial before the previous shift register length change. The test $2L_m \leq m$ must be met before $\sigma^{(m)}(z)$ is updated in the recursion. The discrepancy $d_r$ is the difference between the desired next value $s_{r+k}$ and the actual value computed by the approximating shift register of length $L_r$ having connection polynomial $\sigma^{(r)}(z)$. The recursion must be continued until the minimum-length shift register that generates the error syndrome sequence $s_k, s_{k+1}, \ldots, s_{k+\nu}, \ldots, s_{k+2\nu}$ has been determined, which requires processing all n-k syndrome values to ensure proper termination. The corresponding shift register length is equal to $\nu$, which equals the number of errors that occurred, and the connection polynomial satisfies equations (20) and (21) uniquely [8].

After the LFSR has been synthesized by this algorithm, it is necessary only to continue its operation, with zero input, for an additional k shifts in order to extrapolate the k unknown values of the inverse error transform e(z) [2]. These values are subtracted from the corresponding values of r(z) in order to decode the correct message. This represents a substantial savings in finite-field computation in comparison with the error-value computation of the BCH decoding algorithm.

In the BCH decoding algorithm, after the error locations have been determined, it is necessary to find the error values in order to correct the received sequence and decode the message. The conventional (non-transform) application of the BCH decoding algorithm would require us first to find the error locations by searching for the roots of the error locator polynomial and then to calculate the error values by solving a set of linear equations. Alternatively, the error value calculation can be carried out by evaluating the residues of a partial fraction expansion in the error location singularities of an error-syndrome generating function.

From the transform viewpoint such a procedure is overly complicated. Instead, the LFSR that is characterized by the error locator polynomial is used to extrapolate the inverse transform of the error sequence that was added to the transmitted codeword sequence. Since the message polynomial is the inverse transform of the channel code sequence, we need only to subtract the inverse error transform from the interpolated received sequence to complete the decoding, as was shown in Figure 2.

4.2.2 Modification For Correction Of Errors and Erasures

The decoding algorithm described above was concerned only with correcting errors, an error being described as a received symbol of incorrect (or unknown) value occurring at an unknown location. An erasure is described as an unknown symbol value occurring at a known location; it may occur for example by observing the channel noise, assigning an erasure when the detection decision becomes sufficiently uncertain. A Reed-Solomon code can correct twice as many erasures as errors; in fact it can correct any pattern of t errors and s erasures provided the inequality of eq. (2) is satisfied. A useful Reed-Solomon decoder should be capable of correcting both errors and erasures, which requires some modification to the decoding algorithm used for correcting errors only.

Forney has described a modified BCH decoding algorithm for correcting errors and erasures [17]. In his method the known error locations are used to linearly transform the error syndrome values to a modified syndrome, the latter being used to solve for the error locations. This can be done, for example, by using the modified syndrome as input to the Berlekamp-Massey algorithm for determining the error locator polynomial, followed by a root search. From the known errata locations, a set of linear equations can be solved to determine the unknown symbol values.

In our method of correcting for errors and erasures we initialize the error locator algorithm (Berlekamp-Massey) with the connection polynomial computed from the known erasure locations. Then, we continue the algorithm normally to synthesize an errata locator polynomial which is the product of the error locator polynomial and the erasure locator polynomial. Once the errata locator polynomial is synthesized, there is no further distinction between errors and erasures, and the inverse transform of the errata pattern may be extrapolated by free-running the synthesized LFSR as before.

The erasure locator polynomial $\gamma(z)$ will be defined as

$$\gamma(z) = \prod_{i=1}^{\rho} (z - \tilde{X}_i) = \gamma_\rho + \gamma_{\rho-1}z + \gamma_{\rho-2}z^2 + \ldots + z^\rho \qquad (23)$$

where $\rho$ erasures have occurred, not exceeding the minimum-distance bound of equation (2). The roots $\tilde{X}_i$ designate the erasure locations, forming a set that is disjoint from the error locations $X_i$. It is convenient to define an errata locator polynomial $\tilde{\sigma}(z)$ as the product of the error locator and erasure locator polynomials

$$\tilde{\sigma}(z) = \gamma(z)\sigma(z) = \prod_{i=1}^{\rho} (z-\tilde{X}_i) \prod_{j=1}^{\nu} (z-X_j) \qquad (24)$$

a root of $\tilde{\gamma}(z)$ designating either an error or an erasure location, in other words an errata location.

The erasure locator polynomial can be calculated recursively by the formula

$$\gamma^{(r+1)}(z) = z\gamma^{r}(z) - \tilde{X}_{r+1}\gamma^{r}(z), \qquad \gamma^{(0)} = 1. \tag{25}$$

By comparison of this expression with eq. (19) it is evident that the erasure locator polynomial can be calculated by the error locator's weight calculator if the erasure locations $\tilde{X}_i$ are taken in sequence in place of the next discrepancy values $d_r$.* This initialization sets up the erasure locator $\gamma(z)$ as the characteristic polynomial of the correlation register. During the initialization the first $\rho$ syndrome values, equal to the number of erasures, are sequenced into the correlation register. In the absence of erasures, the initialization reverts to the error-locator mode.

After the apparatus has been initialized with the erasure locator, the algorithm continues in the error locator mode to generate the errata locator polynomial

$$\tilde{\sigma}^{(r+1)}(z) = z^{r-m}\tilde{\sigma}^{(r)}(z) - d_r d_m^{-1}\tilde{\sigma}^{(m)}(z) \tag{26}$$

$$d_r = s_{r+k} + \sum_{i=1}^{L_r} \tilde{\sigma}_i^{(r)} s_{r+k-i} \tag{27}$$

with the initialization at $r = \rho$ given by

$$\tilde{\sigma}^{(\rho)}(z) = \gamma(z) \qquad \tilde{\sigma}^{(m)}(z) = \tilde{\sigma}^{(\rho-1)}(z) = \gamma(z) \tag{28}$$

$$d_\rho = S_{\rho+k} + \sum_{i=1}^{\rho} \gamma_i S_{\rho+k-i} \qquad d_m = d_{\rho-1} = 1$$

*See Figure 8 of section 5.1.

If we write

$$\tilde{\sigma}^{(\rho)}(z) = \gamma(z)\sigma^{(0)}(z)$$
$$\sigma^{(\rho-1)}(z) = \gamma^{(z)}\sigma^{(-1)}(z) \qquad (29)$$

and observe that the errata locator recursion begins at $r = \rho$, we can factor the right hand side of eq. (26) to obtain

$$\tilde{\sigma}^{(r+1)}(z) = \gamma(z)\left[z^{r-m}\sigma^{(r)}(z) - d_r d_m^{-1}\sigma^{(m)}(z)\right] \qquad (30)$$
$$= \gamma(z)\sigma(z)$$

and observe that the term in brackets is the recursion that calculates the error-locator tap weights. If the minimum distance bound of the code has not been exceeded, $2\nu + \rho \leq d - 1$, then the remaining $n-k-\rho$ syndrome digits are sufficient to uniquely determine the error locator factor, $\sigma(z)$, of the errata locator polynomial. This result, as in the case of Massey's algorithm for the error locator, can be proved by induction. Extrapolation of the errata transform from the synthesized errata location shift register implicitly follows from the error-extrapolation proof in Appendix A of reference [2].

### 4.2.3 Continued Fraction Algorithm

Recently a method of continued fractions has been advocated for use as an equivalent to the Berlekamp-Massey algorithm for decoding Reed-Solomon codes [16], As part of our work, we examined this approach, but our analysis indicates slightly reduced computational complexity of the Berlekamp-Massey algorithm, and its convenience of hardware implementation, which we continue to prefer. A summary of our examination of the continued-fraction method is given below.

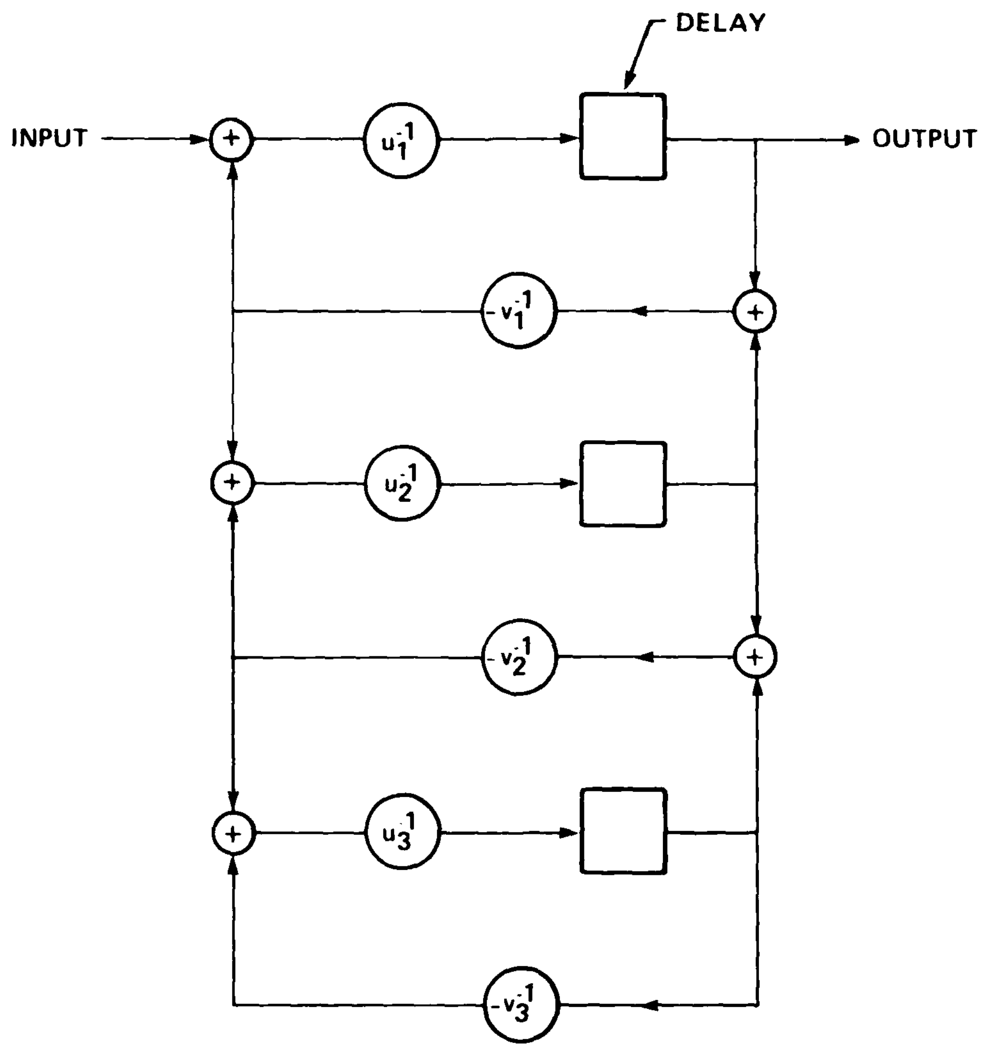
If the error syndrome is expressed as a power series

$$s(z^{-1}) = s_k z^{-1} + s_{k+1} z^{-2} + \ldots + s_{n-1} z^{-(n-k)} + \ldots \qquad (31)$$

it is possible by successive polynomial division to exactly approximate the series by the finite Stieltjes continued-fraction,

$$s(z^{-1}) = \cfrac{1}{u_1 z + \cfrac{1}{v_1 + \cfrac{1}{u_2 z + \cfrac{1}{v_2 + \cdots + \cfrac{1}{u_\nu z + \cfrac{1}{v_\nu}}}}}} \qquad (32)$$

The process of successive division terminates after $2\nu$ steps where $\nu$ is the weight of the error pattern. The continued-fraction expression can be used directly to synthesize the canonic ladder realization of a digital filter network, as shown in Figure 6. The unit sample response of this filter generates (periodically) the inverse transform $e(z)$ of the channel error sequence $E(z)$, which accomplishes the same function as free-running the LFSR synthesized by the Berlekamp-Massey algorithm. In fact, it can be shown that the companion matrix of the ladder network of Figure 6 and the corresponding



IA-61,283

Figure 6. Canonic Ladder for Stieltjes Continued Fraction Realization ($\nu$=3)

LFSR have the same characteristic polynomial $\sigma(z)$.

Instead of using the ladder network we could also use the Stieltjes continued-fraction convergent to form successive approximations to the error locator polynomial, by the iterative set of computations

$$
\begin{aligned}
\sigma^{(0)}(z) &= 1 \\
\sigma^{(1)}(z) &= u_1 z \\
\sigma^{(2)}(z) &= v_1\sigma^{(1)}(z) + \sigma^{(0)}(z) \\
\sigma^{(3)}(z) &= u_2\sigma^{(2)}(z) + \sigma^{(1)}(z) \qquad (33) \\
&\cdot \\
&\cdot \\
&\cdot \\
\sigma^{(2\nu)}(z) &= v_\nu\sigma^{(2\nu-1)}(z) + \sigma^{(2\nu-2)}(z) = \sigma(z)
\end{aligned}
$$

which terminates with the last step of the successive division. The final iteration produces the connection polynomial of the LFSR used to extrapolate the inverse error transform.

In comparing the computational complexity of the Berlekamp-Massey algorithm with the method of continued fractions, observe that both methods in effect synthesize a recursive filter determined by the error locator polynomial. Once this filter is synthesized, the same number of computational steps are required to extrapolate the error transform, so the relative complexity can be compared for the filter synthesis step of the decoding algorithm. The complexity of the two methods, measured by the number of finite-field products computed is relatively equal, the Berlekamp-Massey algorithm always providing the slightly lower value. But there is an important difference for hardware implementation. The Berlekamp-Massey algorithm operates on the syndrome values sequentially, allowing for serial computation of the syndrome.

The continued-fraction algorithm performs a polynomial division involving the error syndrome values at each step of the recursion, thus the syndrome computation must be complete before the continued-fraction algorithm can proceed. An advantage of the continued fraction method is that the number of recursions never exceeds the number of errors being corrected. For the hardware design of our Reed-Solomon decoder we prefer the serial computation of the syndrome.

## 4.3 Message Extraction

Following synthesis of the error (or errata) locator polynomial, the inverse error transform is produced as the unforced response of the synthesized LFSR and the message is extracted by subtracting the inverse error transform from the inverse transform of the received channel sequence. Proof that the error locator polynomial may be used to generate the inverse error transform may be found in Appendix A of reference [2]. The proof relies on two lemmas which can be used to establish the uniqueness of the linear recursion between the inverse error transform and the error locator polynomial, and the sufficiency of the error syndrome to uniquely define the error locator. The reader is referred to the reference for the details of the proof.

## SECTION V

## HARDWARE DESIGN (SUMMARY)

It has consistently been our goal in this work to produce a design, suitable for large-scale circuit integration, that accommodates many Reed-Solomon code parameters in order to maximize the utility of the hardware. A number of design-related tradeoff studies were performed to resolve the issues of hardware programmability, functional partitioning, speed versus gate complexity, interface and control. This work successfully culminated in the complete design, at a basic logic level, of a versatile decoder and in the hardware implementation of a breadboard, the complete details of which are reported separately in Volume II of this report [4]. Salient features of our hardware design are described below.

### 5.1 Functional Partitioning and Programmability

The two principal processing functions used in our Reed-Solomon decoder, the Transformer, which is used both for code generation and error syndrome computation, and the Error Locator, which is used both to compute the error locator polynomial by means of the Berlekamp-Massey LFSR algorithm and to extrapolate the inverse transform of the channel error sequence for use in extracting the corrected message symbols, are described schematically in Figure 7 and Figure 8. They have both been designed to incorporate programmability to accommodate a wide range of code parameters. The total range of Reed-Solomon codes that our design can implement is shown in Table IV. From this table, t the number of correctable errors is determined as one-half the redundancy, or $(\frac{n-k}{2})$. There are 588 separate codes identified by the code parameters (n,k,t) that can be decoded. The subset of multiple-error correcting codes of rate approximately one-half that can be decoded is shown in Table V; there are 17 of these.

IA-61,345

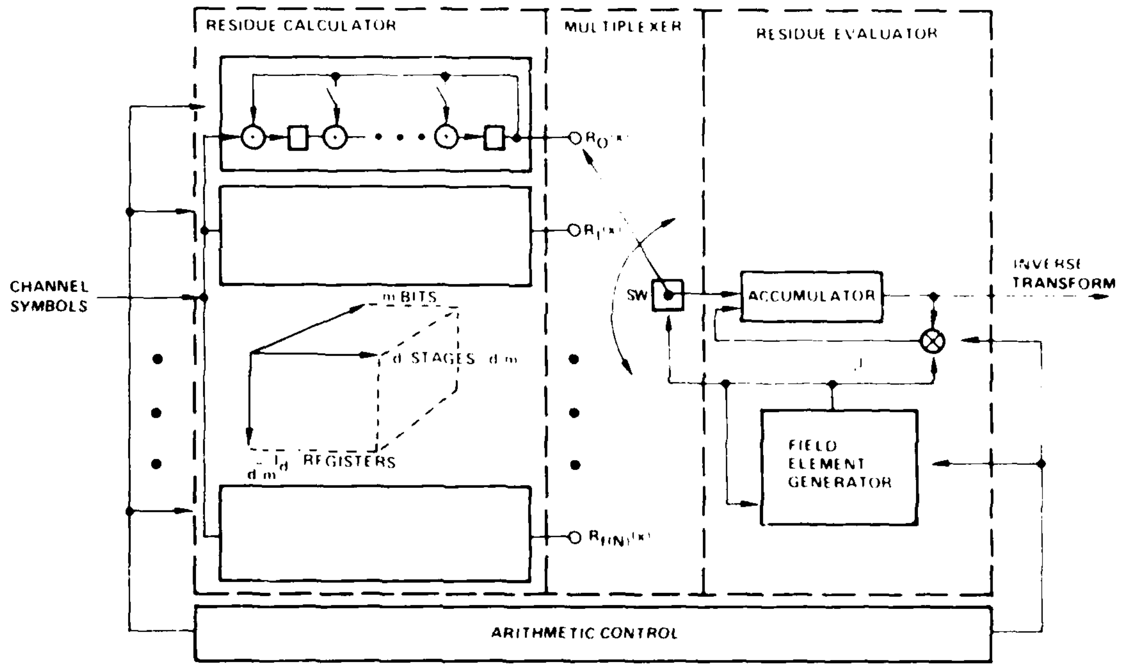


Figure 7. Transform Section

IA-61,346

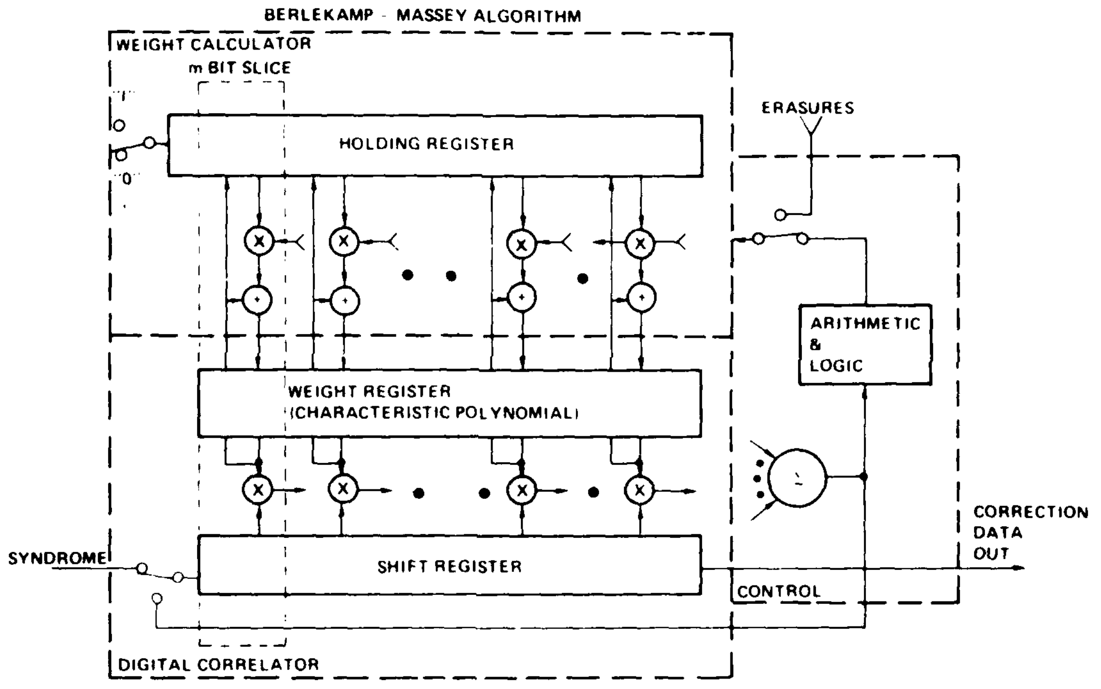


Figure 8. Error Locator

TABLE IV

Reed-Solomon Codes Accommodated by the Hardware Design

| MAXIMUM BLOCK LENGTH (SYMBOLS) | BITS/SYMBOL m | (n, k) CODES $n-k \leq 128$ | |
|---|---|---|---|
| 255 (17 x 5 x 3) | 8 | (255, k)<br>( 85, k)<br>( 51, k)<br>( 17, k)<br>( 15, k)<br>( 5, k)<br>( 3, k) | 304 Total Codes |
| 127 (PRIME) | 7 | (127, k) | 127 Total |
| 63 (7 x 3 x 3) | 6 | ( 63, k)<br>( 21, k)<br>( 9, k)<br>( 7, k)<br>( 3, k) | 103 Total |
| 31 (PRIME) | 5 | ( 31, k) | 31 Total |
| 15 (3 x 5) | 4 | ( 15, k)<br>( 5, k)<br>( 3, k) | 23 Total |

Table V: Half-Rate Codes Accommodated by the Hardware Design

| (n,k) code | Bits/Symbol |
|---|---|
| (255, 127) | 8 |
| (127, 63) | 7 |
| (85, 42) | 8 |
| (63, 31) | 6 |
| (51, 25) | 8 |
| (31, 15) | 5 |
| (21, 10) | 6 |
| (17, 8) | 8 |
| (15, 7) | 8,4 |
| (9, 4) | 6 |
| (7, 3) | 6 |
| (5, 2) | 8,4 |
| (3, 1) | 8,6,4 |

Programmability of the hardware is achieved by stored program control of electronically reconfigurable feedback shift registers, transversal cross-correlators, and finite-field multipliers. For the transformer circuit of Figure 7, the field elements required by the programmable multiplier-accumulator that computes the residues of the divider outputs is generated by a programmable linear feedback shift register. The polynomial division registers, which divide the incoming channel sequence by the minimal polynomials of the field elements and store the remainders, are also configured as binary programmable linear feedback shift registers, both the feedback connections and lengths being reconfigurable by external program control. Notice that the division registers shown schematically in the figure actually represent a set of m identical binary feedback shift registers, each operating independently on a component of the m-bit input symbol. It is expected that the entire set of logic functions for the transformer could be fabricated on a single VLSI (or VHSIC) integrated circuit, the processor complexity requiring about 2000 shift register stages and about 9000 logic gates. The program control would most likely be on a separate control chip containing a modest amount of programmable read-only memory, easily attainable with today's LSI design rules. Custom LSI implementation of the transformer might require several chips.

The error locator circuit, used also to generate the message correction symbols, has a transversal structure that can be either partitioned or fabricated in identical slices as indicated in the dashed portion of Figure 8 for the <u>Weight Calculator</u> and <u>Digital - Correlator</u> sections. The digital correlator section can be thought of as a programmable transversal filter configured as a cross-correlator (or convolver), the syndrome digits providing one set of inputs and the coefficients of the characteristic polynomial providing the other. The weight calculator performs the successive approximations of the characteristic (error locator) polynomial and is also used to initialize the digital correlator by calculating the erasure polynomial recursively from the known erasure locations. The remaining section combines partial vector products produced by the digital correlator section, performs the finite-field residue-class reduction required, and provides input and control functions for the weight calculator and digital correlator sections.

The error locator circuit complexity required for a rate one-half code of block length 255 would require about 4500 shift register stages and about 1400 logic gates. Because of the slice fabrication, we would expect this level of complexity to be attainable on a single VHSIC integrated circuit chip. For custom LSI one might expect to be able to fabricate a 16-symbol section on one substrate; this would require 8 sections for the same code, plus an additional LSI chip for the arithmetic and logic sections. One each of these custom LSI circuits would accommodate a Reed-Solomon (31,15) code; an additional small number of custom LSI transformer chips would be required to complete the decoder.

## 5.2 Throughput

For the design described, if the error locator uses digital logic that operates at the conservative (medium-scale logic) clock rate of 10 MHz; then it takes about 2 μsec to correct a message symbol, or 2Nμsec to decode an N-symbol codeword. For rate one-half codes, there is no output for about one-half the time while the error locator polynomial is being synthesized. During the remaining half of the block duration, the symbols are corrected sequentially. This dead-time could be used effectively with a pair of decoders processing alternate code blocks. For the conservative 10 MHz example, this would permit a constant message throughput of 500K symbols (m-bits) per second; decoding times for rate one-half codes are shown in table VI. The higher clock rates expected for LSI and VHSIC technology

Table VI

Decoding Times for Half-Rate Reed-Solomon Codes

| Code | Bits/Symbol | Block Decoding Time* |
|---|---|---|
| (255, 127) | 8 | 1,020 μsec |
| (127, 63) | 7 | 508 μsec |
| (85, 42) | 8 | 340 μsec |
| (63, 31) | 6 | 252 μsec |
| (51, 25) | 8 | 204 μsec |
| (31, 15) | 5 | 124 μsec |
| (21, 10) | 6 | 84 μsec |
| (17, 8) | 8 | 68 μsec |
| (15, 7) | 8, 4 | 60 μsec |

*Assuming 10 MHz Logic

would permit a proportionately higher throughput. The medium-scale logic breadboard designed and constructed under this project will decode a Reed-Solomon (31,15) code completely in 124 μsec with 50% efficiency.

## REFERENCES

1. Reed, I. S. and Solomon, G., "Polynomial Codes over Certain Finite Fields," J. Soc. Ind. Appl. Math., Vol. 8, pp. 300-304, June 1960.

2. Murakami, H., Reed, I.S., and Welch, L. R., "A Transform Decoder for Reed-Solomon Codes in Multiple-User Communications Systems," IEEE Transactions on Information Theory, Vol. 23, pp. 675-682, November 1977.

3. Blahut, R. E., "Transform Techniques for Error Control Codes," IBM J. Res. Development, Vol. 23, No. 3, May 1979.

4. Carhoun, D. O., Johnson, B. L., and Meehan, S. J., "Transform Decoding of Reed-Solomon Codes: Volume II - Logical Design and Implementation," ESD-TR-82-403, November 1982. (In process)

5. Mandelbaum, D., "Construction of Error Correcting Codes by Interpolation," IEEE Trans. Info. Theory, Vol. I, pp. 27-35, January 1979.

6. Berlekamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York, 1968.

7. Peterson, W., and Weldon, N., Error Correcting Codes, 2nd ed., The MIT PRess, Cambridge, MA, 1972.

8. Massey, J. L., "Shift-Register Synthesis and BCH Decoding," IEEE Trans. Info. Theory, IT-15, No. 1, pp. 122-127, January 1969.

9. Mattson, H. F., and Solomon, G., "A Treatment of Bose-Chaudhuri Codes," J. Soc. Ind. Appl. Math., Vol. 9, pp. 654-669, December 1961.

10. Stone, J. J., "Multiple-Burst Error Correction with the Chinese Remainder Theorem," J. Soc. Ind. Appl. Math., Vol. II, No. 1, March 1963.

11. Tzeng, K. K. and Zimmermann, K. P., "LaGrange's Interpolation Formula and Generalized Goppa Codes," presented at IEEE International Symposium on Information Theory, Ronneby, Sweden, June 1976.

12. Gore, W., "Transmitting Binary Symbols with Reed-Solomon Codes," Johns Hopkins E. E. Report, No. 73-75, Baltimore, MD, April 1973.

13. Michelson, A., "A Fast Transform in Galois Fields and an Application to Decoding Reed-Solomon Codes," presented at IEEE International Symposium on Information Theory, Ronneby, Sweden, June 1976.

14. Pollard, J. M., "The Fast Fourier Transform in a Finite Field," Math. Comput., Vol. 25, pp. 365-374, April 1971.

15. Reed, I. S., Truong, T. K., Miller, R. L., and Huang, J. P., "Fast Transforms for Decoding Reed-Solomon Codes," Proc. IEEE, 1980 (to be published).

16. Reed, I. S., Scholtz, R. A., Truong, T. K., and Welch, L. R., "The Fast Decoding of Reed-Solomon Codes Using Fermat Theoretic Transforms and Continued Fractions," IEEE Trans. on Info. Theory, IT-24, pp. 100-106, 1978.

17. Forney, G. D., Jr., Concatenated Codes, MIT Press, Cambridge, MA, 1966.